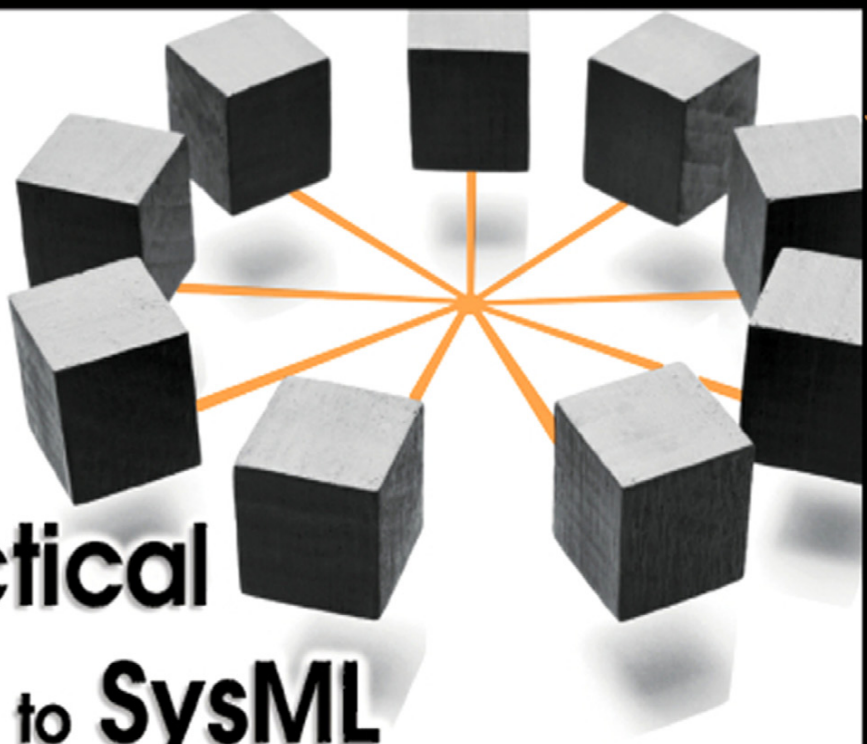
SECOND EDITION

The MK/OMG PRESS



A Practical Guide to SysML

The Systems Modeling Language

Sanford Friedenthal
Alan Moore
Rick Steiner

MK MORGAN KAUFMANN

OMG OBJECT MANAGEMENT GROUP

OBJECT MANAGEMENT GROUP SM CERTIFICATION PROGRAM

CHAPTER

Model-Based Systems Engineering

2

Model-based systems engineering (MBSE) applies systems modeling as part of the systems engineering process described in Chapter 1 to support analysis, specification, design, and verification of the system being developed. A primary artifact of MBSE is a coherent model of the system being developed. This approach enhances specification and design quality, reuse of system specification and design artifacts, and communications among the development team.

This chapter summarizes MBSE concepts to provide further context for SysML without emphasizing a specific modeling language, method, or tool. MBSE is contrasted with the more traditional document-based approach to motivate the use of MBSE and highlight its benefits. Principles for effective modeling are also discussed.

2.1 CONTRASTING THE DOCUMENT-BASED AND MODEL-BASED APPROACH

The following sections contrast the document-based approach and the model-based approach to systems engineering.

2.1.1 Document-Based Systems Engineering Approach

Traditionally, large projects have employed a **document-based systems engineering** approach to perform the systems engineering activities referred to in Chapter 1, Section 1.2. This approach is characterized by the generation of textual specifications and design documents, in hard-copy or electronic file format, that are then exchanged between customers, users, developers, and testers. System requirements and design information are expressed in these documents and drawings. The systems engineering emphasis is placed on controlling the documentation and ensuring the documents and drawings are valid, complete, and consistent, and that the developed system complies with the documentation.

In the document-based approach, specifications for a particular system, its subsystems, and its hardware and software components are usually depicted in a hierarchical tree, called a **specification tree.** A **systems engineering management plan** (**SEMP)** documents how the systems engineering process is employed on the project, and how the engineering disciplines work together to develop the documentation needed to satisfy the requirements in the specification tree. Systems engineering activities are planned by estimating the time and effort required to generate documentation, and progress is then measured by the state of completion of the documents.

Document-based systems engineering typically relies on a concept of operation document to define how the system is used to support the required mission or objective. Functional analysis is performed

A Practical Guide to SysML. DOI: 10.1016/B978-0-12-385206-9.00002-8

to decompose the system functions, and allocate them to the components of the system. Drawing tools are used to capture the system design, such as functional flow diagrams and schematic block diagrams. These diagrams are stored as separate files and included in the system design documentation. Engineering trade studies and analyses are performed and documented by many different disciplines to evaluate and optimize alternative designs and allocate performance requirements. The analysis may be supported by individual analysis models for performance, reliability, safety, mass properties, and other aspects of the system.

Requirements traceability is established and maintained in the document-based approach by tracing requirements between the specifications at different levels of the specification hierarchy. Requirements management tools are used to parse requirements contained in the specification documents and capture them in a requirements database. The traceability between requirements and design is maintained by identifying the part of the system or subsystem that satisfies the requirement, and/or the verification procedures used to verify the requirement, and then reflecting this in the requirements database.

The document-based approach can be rigorous but has some fundamental limitations. The completeness, consistency, and relationships between requirements, design, engineering analysis, and test information are difficult to assess since this information is spread across several documents. This makes it difficult to understand a particular aspect of the system and to perform the necessary traceability and change impact assessments. This, in turn, leads to poor synchronization between system-level requirements and design and lower-level hardware and software design. It also makes it difficult to maintain or reuse the system requirements and design information for an evolving or variant system design. Also, progress of the systems engineering effort is based on the documentation status, which may not adequately reflect the quality of the system requirements and design. These limitations can result in inefficiencies and potential quality issues that often show up during integration and testing, or worse, after the system is delivered to the customer.

2.1.2 Model-Based Systems Engineering Approach

A model-based approach has been standard practice in electrical and mechanical design and other disciplines for many years. Mechanical engineering transitioned from the drawing board to increasingly more sophisticated two-dimensional (2D) and then three-dimensional (3D) computer-aided design tools beginning in the 1980s. Electrical engineering transitioned from manual circuit design to automated schematic capture and circuit analysis in a similar time-frame. Computer-aided software engineering became popular in the 1980s for using graphical models to represent software at abstraction levels above the programming language. The use of modeling for software development is becoming more widely adopted, particularly since the advent of the Unified Modeling Language in the 1990s.

The model-based approach is becoming more prevalent in systems engineering. A mathematical formalism for MBSE was introduced in 1993 by Wayne Wymore [31]. The increasing capability of computer processing, storage, and network technology along with emphasis on systems engineering standards has created an opportunity to significantly advance the state of the practice of MBSE. It is expected that MBSE will become standard practice in a similar way that it has with other engineering disciplines.

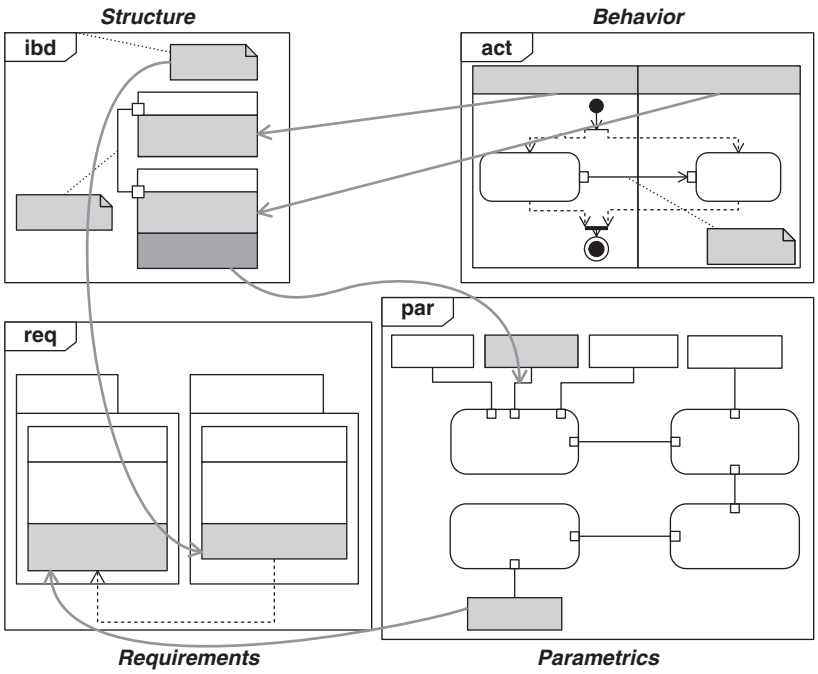


FIGURE 2.1

Representative system model example in SysML. (Specific model elements have been deliberately obscured and will be discussed in subsequent chapters.)

"**Model-based systems engineering (MBSE)** is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [32]. MBSE is intended to facilitate systems engineering activities that have traditionally been performed using the document-based approach and result in enhanced specification and design quality, reuse of system specification and design artifacts, and communications among the development team. The output of the systems engineering activities is a coherent model of the system (i.e., system model), where the emphasis is placed on evolving and refining the model using model-based methods and tools.

The System Model

The **system model** is generally created using a modeling tool and contained in a model repository. The system model includes system specification, design, analysis, and verification information. The model consists of model elements that represent requirements, design, test cases, design rationale, and their interrelationships. Figure 2.1 shows the system model as an interconnected set of model elements that represent key system aspects as defined in SysML, including its structure, behavior, parametrics, and requirements. The multiple cross-cutting relationships between the model elements contained in the

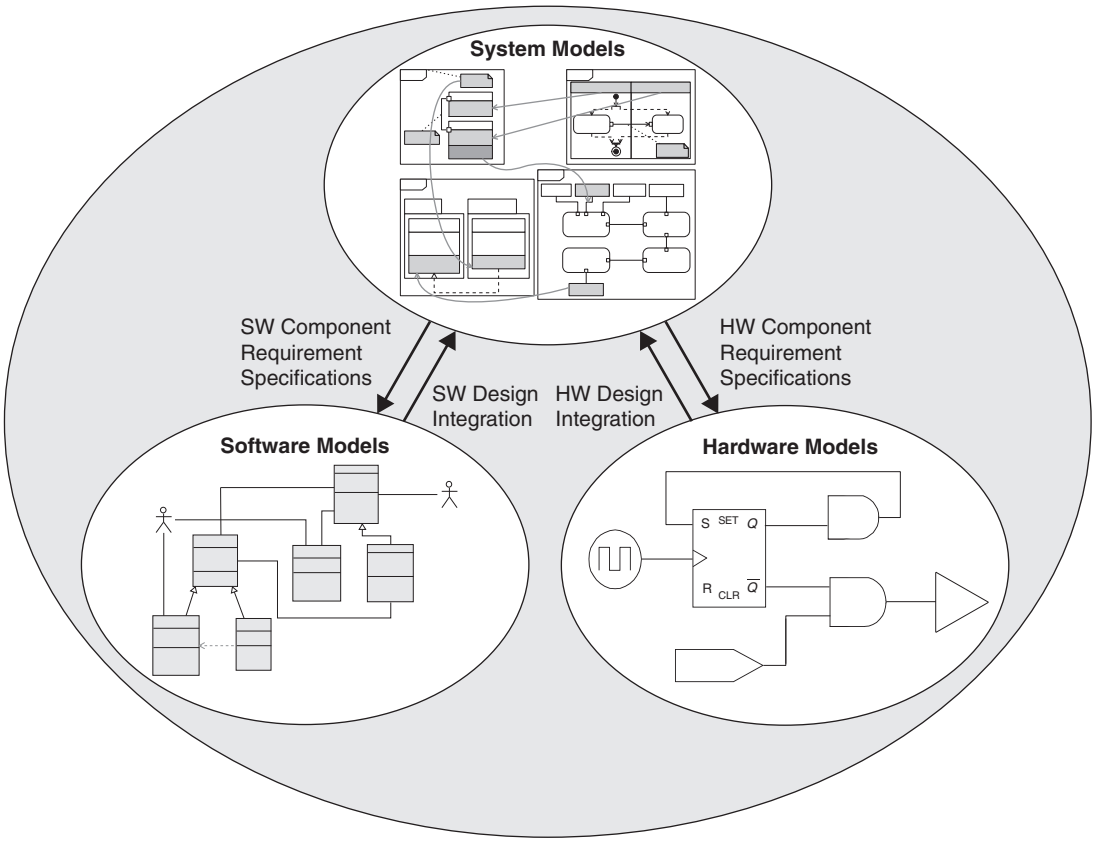


FIGURE 2.2

The system model is used to specify the components of the system.

model repository enable the system model to be viewed from many different perspectives to focus on different aspects of the system.

A primary use of the system model is to enable the design of a system that satisfies its requirements and supports the allocation of the requirements to the system's components. Figure 2.2 depicts how the system model is used to specify the components of the system. The system model includes component interconnections and interfaces, component interactions and the associated functions the components must perform, and component performance and physical characteristics. The textual requirements for the components may also be captured in the model and traced to system requirements.

In this regard, the system model is used to specify the component requirements and can be used as an agreement between the system designer and the subsystem and/or component developer. The component developers receive the component requirements in a way that is meaningful to them either through a model data exchange mechanism or by providing documentation that is automatically generated from the model. The component developer can provide information about how the

component design satisfies its requirements in a similar way. The use of a system model provides a mechanism to specify and integrate subsystems and components into the system, and maintain traceability between system and component requirements.

The system model can also be integrated with engineering analysis and simulation models to perform computation and dynamic execution. If the system model is to be executed directly, the system modeling environment must be augmented with an execution environment. A discussion of model execution is included in Chapter 18, Section 18.1.3.

The Model Repository

The model elements that compose the system model are stored in a **model repository** and depicted on diagrams by graphical symbols. The modeling tool enables the modeler to create, modify, and delete individual model elements and their relationships in the model repository. The modeler uses the symbols on the diagrams to enter the model information into the model repository and to view model information from the repository. The specification, design, analysis, and verification information previously captured in documents is now captured in the model repository. The model can be viewed in diagrams or tables or in reports generated by querying the model repository. The views enable understanding and analysis of different aspects of the same system model. The documents can continue to serve as an effective means for reporting the information, but in MBSE, the text and graphical information contained in documentation is generated from the model. In fact, many of the modeling tools have a flexible and automated document-generation capability that can significantly reduce the time and cost of building and maintaining the system specification and design documentation.

Model elements corresponding to requirements, design, analysis, and verification information are traceable to one another through their relationships, even if they are represented on different diagrams. For example, an engine component in an automobile system model may have many relationships to other elements in the model. It is part of the automobile system, connected to the transmission, satisfies a power requirement, performs a function to convert fuel to mechanical energy, and has a weight property that contributes to the vehicle's weight. These relationships are part of the system model.

The modeling language imposes rules that constrain which relationships can exist. For example, the model should not allow a requirement to contain a system component or an activity to produce inputs instead of outputs. Additional model constraints may be imposed based on the method being employed. An example of a method-imposed constraint may be that all system functions must be decomposed and allocated to a component of the system. Modeling tools are expected to enforce constraints at the time the model is constructed, or by running a model-checker routine at the modeler's convenience and providing a report of the constraint violations.

The model provides much finer-grain control of the information than is available in a document-based approach, where this information may be spread across many documents and the relationships may not be explicitly defined. The model-based approach promotes rigor in the specification, design, analysis, and verification process. It also significantly enhances the quality and timeliness of traceability and impact assessment over the document-based approach.

Systems engineering is intended to fit the pieces of the system together into a cohesive whole. MBSE must support this fundamental focus of systems engineering. In particular, a growing emphasis for the system model is its role in providing an integration framework for models created by other engineering disciplines, including hardware, software, test, and many specialty engineering disciplines

such as reliability, safety, and security. This is discussed in Chapter 18, as part of the overall discussion on integrating the system model into a Systems Development Environment.

Transitioning to MBSE

Models and related diagramming techniques have been used as part of the document-based systems engineering approach for many years, and include functional flow diagrams, behavior diagrams, schematic block diagrams, N2 charts, performance simulations, and reliability models, to name a few. However, the use of models has generally been limited in scope to support specific types of analysis or selected aspects of system design. The individual models have not been integrated into a coherent model of the overall system, and the modeling activities have not been integrated into the systems engineering process. The transition from document-based systems engineering to MBSE is a shift in emphasis from controlling the documentation about the system to controlling the model of the system. MBSE integrates system requirements, design, analysis, and verification models to address multiple aspects of the system in a cohesive manner, rather than a disparate collection of individual models.

MBSE provides an opportunity to address many of the limitations of the document-based approach by providing a more rigorous means for capturing and integrating system requirements, design, analysis, and verification information, and facilitating the maintenance, assessment, and communication of this information across the system's life cycle. Some of the MBSE potential benefits include the following:

- Enhanced communications
  - Shared understanding of the system across the development team and other stakeholders
  - Ability to integrate views of the system from multiple perspectives
- Reduced development risk
  - Ongoing requirements validation and design verification
  - More accurate cost estimates to develop the system
- Improved quality
  - More complete, unambiguous, and verifiable requirements
  - More rigorous traceability between requirements, design, analysis, and testing
  - Enhanced design integrity
- Increased productivity
  - Faster impact analysis of requirements and design changes
  - More effective exploration of trade-space
  - Reuse of existing models to support design evolution
  - Reduced errors and time during integration and testing
  - Automated document generation
- Leveraging the models across life cycle
  - Support operator training on the use of the system
  - Support diagnostics and maintenance of the system
- Enhanced knowledge transfer
  - Capture of existing and legacy designs
  - Efficient access and modification of the information

MBSE can provide additional rigor in the specification and design process when implemented using appropriate methods and tools. However, this rigor does not come without a price. Clearly,

transitioning to MBSE underscores the need for up-front investment in processes, methods, tools, and training. It is expected that during the transition, MBSE will be performed in combination with document-based approaches. For example, the upgrade of a large, complex legacy system still relies heavily on the legacy documentation, and only parts of the system may be modeled. Careful tailoring of the approach and scoping of the modeling effort is essential to meet the needs of a particular project. The considerations for transitioning to MBSE are discussed in Chapter 19.

2.2 MODELING PRINCIPLES

The following sections provide a brief overview of some of the key modeling principles.

2.2.1 Model and MBSE Method Definition

A **model** is a representation of one or more concepts that may be realized in the physical world. It generally describes a **domain of interest.** A key feature of a model is that it is an abstraction that does not contain all the detail of the modeled entities within the domain of interest. Models can be abstract mathematical and logical representations, as well as more concrete physical prototypes. The form of expression for the more abstract representation may be a combination of graphical symbols, such as nodes and arcs on a graph or geometric representations, and text, such as the text statements in a programming language. A common example of a model is a blueprint of a building and a scaled prototype physical model. The building blueprint is a specification for one or more buildings that are built. The blueprint is an abstraction that does not contain all the building's detail, such as the detailed characteristics of its materials.

A system model expressed in SysML is analogous to a building blueprint that specifies a system to be implemented. Instead of a geometric representation of the system, the SysML model represents the behavior, structure, properties, constraints, and requirements of the system. SysML has a semantic foundation that specifies the types of model elements and the relationships that can appear in the system model. The model elements that comprise the system model are stored in a model repository and can be represented graphically. A SysML model can also be simulated if it is supported by an execution environment.

A **method** is a set of related activities, techniques, and conventions that implement one or more processes and is generally supported by a set of tools. A **model-based systems engineering method** is a method that implements all or part of the systems engineering process, and produces a system model as one of its primary artifacts.

2.2.2 The Purpose for Modeling a System

The purpose for modeling a system for a particular project must be clearly defined in terms of the expected results of the modeling effort, the stakeholders who use the results, and how the results are intended to be used. The model purpose is used to determine the scope of the modeling effort in terms of model breadth, depth, and fidelity. This scope should be balanced with the available schedule, budget, skill levels, and other resources. Understanding the purpose and scope provides the basis for establishing realistic expectations for the modeling effort. The purposes for modeling a system may

emphasize different aspects of the systems engineering process or support other life-cycle uses, including the following:

- Characterize an existing system
- Specify and design a new or modified system
  - Represent a system concept
  - Specify and validate system requirements
  - Synthesize system designs
  - Specify component requirements
  - Maintain requirements traceability
- Evaluate the system
  - Conduct system design trade-offs
  - Analyze system performance requirements or other quality attributes
  - Verify that the system design satisfies its requirements
  - Assess the impact of requirements and design changes
  - Estimate the system cost (e.g., development, life cycle)
- Train users on how to operate or maintain a system

2.2.3 Establishing Criteria to Meet the Model Purpose

Criteria can be established to assess how well a model can meet its modeling purpose. However, one must first distinguish between a good model and a good design. One can have a good model of a poor design or a poor model of a good design. A good model meets its intended purpose. A good design is based on how well the design satisfies its requirements and the extent to which it incorporates quality design principles. As an example, one could have a good model of a chair that meets the intended purpose of the model by providing an accurate representation of the chair design. However, the chair's design may be a poor design if it does not have structural integrity. A good model provides visibility to aid the design team in identifying issues and assessing design quality. The selected MBSE method and tools should facilitate a skilled team to develop both a good model and a good design.

The answers to the following questions can be used to assess the effectiveness of the model and derive quality attributes for the model. The quality attributes in turn can be used to establish preferred modeling practices. The modeling tool can be used to check the model quality, such as whether the model is well formed.

Is the Model's Scope Sufficient to Meet Its Purpose?

Assuming the purpose is clearly defined as described earlier, the scope of the model is defined in terms of its breadth, depth, and fidelity. The model scope significantly impacts the level of resources required to support the modeling effort.

Model breadth. The breadth of the model must be sufficient for the purpose by determining which parts of the system need to be modeled, and the extent to which the model addresses the system requirements. This question is particularly relevant to large systems where one may not need to model the entire system to meet project needs. If new functionality is being added to an existing system, one may choose to focus on modeling only those portions needed to support the new functionality. In an automobile design, for example, if the emphasis is on new requirements for fuel

economy and acceleration, the model may focus on elements related to the power train, with less focus on the braking and steering subsystems.

Model depth. The depth of the model must be sufficient for the purpose by determining the level of the system design hierarchy that the model must encompass. For a conceptual design or initial design iteration, the model may only address a high level design. In the automobile example, the initial design iteration may only model the system to the engine black box level, whereas if the engine is subject to further development, a future design iteration may model the engine components.

Model fidelity. The fidelity of the model must be sufficient for the purpose by determining the required level of detail. For example, a low-fidelity behavioral model may be sufficient to communicate a simple ordering of actions in an activity diagram. Additional model detail is required if the behavioral model is intended to be executed, but this additional detail can add to the level of understanding of the system response. As another example, when modeling interfaces, a low-fidelity model may only include the logical interface description, whereas a higher-fidelity model may model the message structure and communication protocol. Finally, further timing information may be required to model system performance.

Is the Model Complete Relative to Its Scope?

A necessary condition for the model to be complete is that its breadth, depth, and fidelity must match its defined scope. Other completeness criteria may relate to other quality attributes of the model (e.g., whether the naming conventions have been properly applied) and design completion criteria (e.g., whether all design elements are traced to a requirement). The MBSE metrics discussed in Section 2.2.4 can be used to establish additional completion criteria.

Is the Model Well Formed Such That Model Constraints Are Adhered to?

A well-formed model conforms to the rules of the modeling language. For example, the rules in SysML do not allow a requirement to contain a system component, although other relationships are allowed between components and requirements such as the satisfy relationship. The modeling tool should enforce the constraints imposed by the rules of the modeling language or provide a report of violations.

Is the Model Consistent?

In SysML, some rules are built into the language to ensure model consistency. For example, compatibility rules can support type checking to determine whether interfaces are compatible or whether units are consistent on different properties. Additional constraints can be imposed by the method used. For example, a method may impose a constraint that logical components can only be allocated to hardware, software, or operational procedures. These constraints can be expressed in the object constraint language (OCL) [33] and enforced by the modeling tool.

Enforcing constraints assists in maintaining consistency across the model, but it does not prevent design inconsistencies. A simple example may be that two modelers inadvertently give the same component two different names that are interpreted by a model checker as different components. This type of inconsistency should readily surface through the reviews and report generation. However, the likelihood of inconsistencies increases when multiple people are working on the same model.

A combination of well-defined model conventions and a disciplined process can reduce the likelihood of this happening.

Is the Model Understandable?

There are many factors driven by the model-based method and modeling style that can contribute to understandability. A key contributing factor to enhance understandability is the effective use of model abstraction. For example, when describing the functionality of an automobile, one could describe a top-level function as "drive car" or provide a more detailed functional description such as "turn ignition on, put gear into drive, push accelerator pedal," and so on. An understandable model should include multiple levels of abstraction that represent different levels of detail but relate to one another. As will be described in later chapters, the use of decomposition, specialization, allocation, views, and other modeling approaches in SysML are used to represent different levels of abstraction.

Another factor that impacts understandability relates to the presentation of information on the diagrams themselves. Often, there is a lot of detail in the model, but only selected information is relevant to communicate a particular aspect of the design. The information on the diagram can be controlled by using the tool capability to elide (hide) nonessential information and display only the information relevant to the diagram's purpose. Again, the goal is to avoid information overload for the reviewer of the model.

Other factors that contribute to understandability are the use of modeling conventions and the extent to which the model is self-documenting as described next.

Are Modeling Conventions Documented and Used Consistently?

Modeling conventions and standards are critical to ensure consistent representation and style across the model. This includes establishing naming conventions for each type of model element, diagram names, and diagram content. Naming conventions may include stylistic aspects of the language, such as when to use uppercase versus lowercase, and when to use spaces in names. The conventions and standards should also account for tool-imposed constraints, such as limitations in the use of alphanumeric and special characters. It is also recommended that a template be established for each diagram type so that consistent style can be applied.

Is the Model Self-documenting in Terms of Providing Sufficient Supporting Information?

The use of annotations and descriptions throughout the model can help to provide value-added information if applied consistently. This can include the rationale for design decisions, flagging issues or problem areas for resolution, and providing additional textual descriptions for model elements. This enables longer-term maintenance of the model and enables it to be more effectively communicated to others.

Does the Model Integrate with Other Models?

The system model may need to integrate with electrical, mechanical, software, test, and engineering analysis models. This capability is determined by the specific method, tool implementation, and modeling languages used. For example, the approach for passing information from the system model using SysML to a software model using UML can be defined for specific methods, tools, and interchange standards. In general, this is addressed by establishing an agreed-on expression of the modeling information so that it can be best communicated to the user of the information, such as hardware and

software developers, testers, and engineering analysts. The approach for integration of models and tools is discussed in Chapter 18.

2.2.4 Model-Based Metrics

Measurement data collection, analysis, and reporting can be used as a management technique throughout the development process to assess design quality and progress. This in turn is used to assess technical, cost, and schedule status and risk, and to support ongoing project planning and control. **Model-based metrics** can provide useful data that can be derived from a system model expressed in SysML to help answer the questions below. The data can be collected over time to provide significant additional insight through assessment of the data trends and statistical distributions.

What Is the Quality of the Design?

Metrics can be defined to measure the quality of a model-based system design. Some of these metrics, such as assessing requirements satisfaction, requirements verification, and technical performance measurement are based on metrics that have been traditionally used in document-centric designs. Other metrics may include indicators, for example, of how well the design is partitioned.

A SysML model can include explicit relationships that can be used to measure the extent to which the requirements are satisfied. The model can provide granularity by identifying model elements that satisfy specific requirements. The requirements traceability can be established from mission-level requirements down to component-level requirements. Other SysML relationships can be used in a similar way to measure which requirements have been verified. This data can be captured directly from the model or indirectly from a requirements management tool that is integrated with the SysML modeling tool.

A SysML model can include critical properties that are monitored throughout the design process. Typical properties may include performance properties, such as latency, physical properties (e.g., weight), and other properties (e.g., reliability and cost). These properties can be monitored using standard technical performance measurement (TPM) techniques. The model can also include parametric relationships among the properties that indicate how they may be impacted as a result of design decisions.

Design partitioning can be measured in terms of the level of cohesion and coupling of the design. Coupling can be measured in terms of the number of interfaces or in terms of more complex measures of dependencies between different model parts. Cohesion metrics are more difficult to define, but measure the extent to which a component can perform its functions without requiring access to external data. The object-oriented concept of encapsulation reflects this concept.

What Is the Progress of the Design and Development Effort?

Model-based metrics can be defined to assess design progress by establishing completion criteria for the design. The quality attributes in the previous section refer to whether the model is complete relative to the defined scope of the modeling effort. This is necessary, but not sufficient, to assess design completeness. The requirements satisfaction described to measure design quality can also be used to assess design completeness. Other metrics may include the number of use case scenarios that have been completed or the percent of logical components that have been allocated to physical components. From a systems engineering perspective, a key measure of system design completeness is

the extent to which components have been specified. This metric can be measured in terms of the completeness of the specification of component interfaces, behavior, and properties.

Other metrics for assessing progress include the extent to which components have been verified and integrated into the system, and the extent to which the system has been verified to satisfy its requirements. Test cases and verification status can be captured in the model and used as a basis for this assessment.

What Is the Estimated Effort to Complete Design and Development?

The Constructive Systems Engineering Cost Model (COSYSMO) is used for estimating the cost and effort to perform systems engineering activities. This model includes both sizing and productivity parameters, where the size estimates the magnitude of the effort, and productivity factors are applied to arrive at an actual labor estimate to do the work.

When using model-based approaches, sizing parameters can be identified in the model in terms of the number of different modeling constructs that may include the following:

Model elements
Requirements
Use cases
Scenarios
States
System and component interfaces
System and component activities or operations
System and component properties
Components by type (e.g., hardware, software, data, operational procedures)
Constraints
Test cases

The metrics should also account for relationships between these model elements, such as the number of requirements that are satisfied, number of requirements that are verified, the number of use cases that are realized, the number of activities that are allocated to blocks, and the number of analyses that have been performed.

The MBSE sizing parameters are integrated into the cost model. The parameters may have complexity factors associated with them as well. For example, the complexity of a use case may be indicated by the number of actors participating in the interaction. Additional factors to be considered are the amount of reuse and modification of existing models, versus creating new models.

Sizing and productivity data need to be collected and validated over time to establish statistically meaningful data and cost estimating relationships to support accurate cost estimating. However, early users of MBSE can identify sizing parameters that contribute most significantly to the modeling effort, and use this data for local estimates and to assess productivity improvements over time.

2.2.5 Other Model-Based Metrics

The previous discussion is a sampling of some of the model-based metrics that can be defined. Many other metrics can also be derived from the model, such as the stability of the number of requirements and design changes over time, or potential defect rates. The metrics can also be derived to establish

benchmarks from which to measure the MBSE benefits as described in Section 2.1.2, such as the productivity improvements resulting from MBSE over time. These metrics should be defined and captured to support the business case for MBSE. Chapter 19, Section 19.1.1 includes a discussion of additional metrics related to deploying MBSE in an organization.

2.3 SUMMARY

The practice of systems engineering is transitioning from a document-based approach to a model-based approach like many other engineering disciplines, such as mechanical and electrical engineering, have already done. MBSE offers significant potential benefits to enhance specification and design quality and consistency, reuse of specification and design artifacts, and communications among the development team, yielding overall improvements in quality, productivity, and reduced development risk. The emphasis for MBSE is to produce and control a coherent system model, and use this model to specify and design the system. Modeling can support many purposes, such as to represent a system concept or specify system components. A good model meets its intended purpose, and the scope of the model should support its purpose within the resource constraints of the modeling effort. Quality attributes of a model, such as model consistency, understandability, and well-formedness, and the use of modeling conventions, can be used to assess the effectiveness of a model and to drive preferred modeling practices. MBSE metrics can be used to assess design quality, progress and risk, and support management of the development effort.

2.4 QUESTIONS

1. What are some of the primary distinctions between MBSE and a document-based approach?
2. What are some of the benefits of MBSE over the document-based approach?
3. Where are the model elements of a system model stored?
4. Which aspects of the model can be used to define the scope of the model?
5. What constitutes an effective model?
6. What are some of the quality attributes of an effective model?
7. What is the difference between a good model and a good design?
8. What are examples of questions that MBSE metrics can help answer?
9. What are possible sizing parameters that could be used to estimate an MBSE effort?

This page intentionally left blank

CHAPTER

Getting Started with SysML

3

This chapter provides an introduction to SysML and guidance on how to begin modeling in SysML. The chapter provides a brief overview of SysML, and then introduces a simplified version of the language we refer to as SysML-Lite, along with a simplified example, and tool tips on how to capture the model in a typical modeling tool. This chapter also introduces a simplified model-based systems engineering (MBSE) method that is consistent with the systems engineering process described in Chapter 1, Section 1.2. The chapter finishes by describing some of the challenges involved in learning SysML and MBSE.

3.1 SysML PURPOSE AND KEY FEATURES

SysML[1] is a general-purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. These systems may include hardware, software, data, personnel, procedures, facilities, and other elements of man-made and natural systems. The language is intended to help specify and architect systems and specify their components that can then be designed using other domain-specific languages such as UML for software design and VHDL and three-dimensional geometric modeling for hardware design. SysML is intended to facilitate the application of an MBSE approach to create a cohesive and consistent model of the system that yields the benefits described in Chapter 2, Section 2.1.2.

SysML can represent the following aspects of systems, components, and other entities:

- Structural composition, interconnection, and classification
- Function-based, message-based, and state-based behavior
- Constraints on the physical and performance properties
- Allocations between behavior, structure, and constraints
- Requirements and their relationship to other requirements, design elements, and test cases

3.2 SysML DIAGRAM OVERVIEW

SysML includes nine diagrams as shown in the diagram taxonomy in Figure 3.1. Each diagram type is summarized here, along with its relationship to UML diagrams:

- *Package diagram* represents the organization of a model in terms of packages that contain model elements (same as UML package diagram)

[1]OMG Systems Modeling Language (OMG SysML™) is the official name of the language, but it is referred to as SysML for short. Additional information on SysML can be found at the official OMG SysML website at http://www.omgsysml.org.

A Practical Guide to SysML. DOI: 10.1016/B978-0-12-385206-9.00003-X

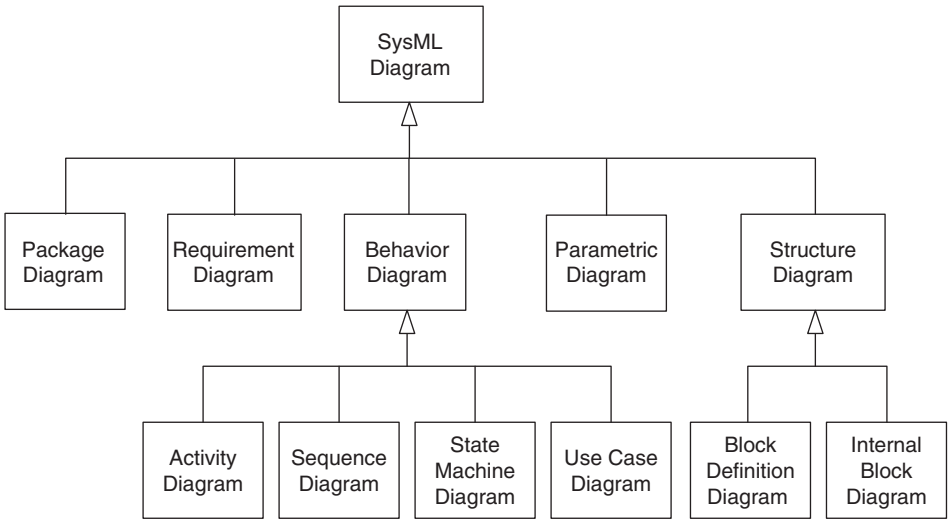


FIGURE 3.1

SysML diagram taxonomy.

- *Requirement diagram* represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability (not in UML)
- *Activity diagram* represents behavior in terms of the order in which actions execute based on the availability of their inputs, outputs, and control, and how the actions transform the inputs to outputs (modification of UML activity diagram)
- *Sequence diagram* represents behavior in terms of a sequence of messages exchanged between systems, or between parts of systems (same as UML sequence diagram)
- *State machine diagram* represents behavior of an entity in terms of its transitions between states triggered by events (same as UML state machine diagram)
- *Use case diagram* represents functionality in terms of how a system is used by external entities (i.e., actors) to accomplish a set of goals (same as UML use case diagram)
- *Block definition diagram* represents structural elements called blocks, and their composition and classification (modification of UML class diagram)
- *Internal block diagram* represents interconnection and interfaces between the parts of a block (modification of UML composite structure diagram)
- *Parametric diagram* represents constraints on property values, such as $F = m * a$, used to support engineering analysis (not in UML)

A diagram graphically represents a particular aspect of the system model as described in Chapter 2, Section 2.1.2. The kinds of model elements and associated symbols (e.g., diagram elements) that can appear on a diagram are constrained by its diagram kind. For example, an activity diagram can include diagram elements that represent actions, control flow, and input/output flow (i.e., object flow), but not diagram elements for connectors and ports. As a result, a diagram represents a subset of the underlying model repository, as described in Chapter 2, Section 2.1.2. Tabular representations, such as allocation

tables, are also supported in SysML as a complement to diagram representations to represent model information.

3.3 INTRODUCING SysML-LITE

SysML-Lite is introduced here as a simplified version of the language to help people get started modeling with SysML, but is not part of the SysML standard. It includes six of the nine SysML diagrams, and a small subset of the available language features for each diagram kind. SysML-Lite provides a significant modeling capability. This section provides a brief introduction to SysML-Lite, along with a simple example to highlight the features of SysML-Lite. This section also includes tool tips to assist a new modeler in the use of a typical modeling tool.

3.3.1 SysML-Lite Diagrams and Language Features

The six (6) kinds of diagrams that are part of SysML-Lite are highlighted in Figure 3.2. Each diagram contains a diagram header that identifies the diagram kind, and other information about the diagram which is explained in Chapter 5, Section 5.3.2. In particular, SysML-Lite includes the:

- package diagram to capture the model organization
- requirement diagram to capture text-based requirements
- activity diagram to represent the behavior of the system and its components
- block definition diagram to represent the system hierarchy
- internal block diagram to represent the system interconnection
- parametric diagram to capture the relationship among system properties to support engineering analysis

This set of diagrams provides a model user with a substantial capability for modeling systems that covers many of the classical systems engineering diagrams and more.

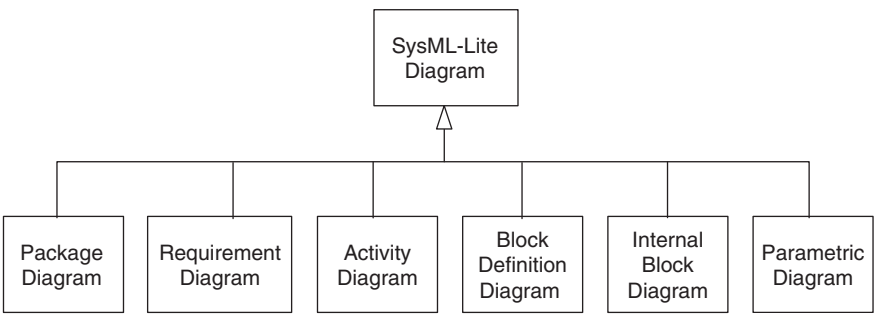


FIGURE 3.2

SysML-Lite includes six of the nine SysML diagrams and a subset of the language features. It is intended to introduce a new modeler to SysML, while providing a substantial modeling capability.

SysML-Lite includes a small subset of the language features for each of the six SysML diagrams. Some of the features of SysML-Lite are represented in the diagrams in Figure 3.3. The precise subset of SysML language features can be adapted to the need. The figure also shows thick lines with arrowheads that are not part of the language, but highlight some of the important cross diagram relationships. These relationships are generally consistent with classical systems engineering methods, such as functional decomposition and allocation.

The package diagram, labeled pkg, is used to organize the **model elements** contained in the model. In this diagram, the *System Model* appears in the diagram header and contains packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. Each of these packages, in turn, contains model elements that are represented on the requirements diagram, activity diagram, block definition diagram, internal block diagram, and parametric diagram, respectively. Note that model elements for both the block definition diagram and internal block diagram are contained in the *Structure* package.

The requirement diagram is labeled req and represents a simple hierarchy of text-based requirements that are typically part of a specification document. The top level requirement named *R1* contains two requirements *R1.1* and *R1.2.* The corresponding requirement statement for *R1.1* is captured as a text property of the requirement and corresponds to the text that would be found for this requirement in the specification document.

The activity diagrams are labeled act. The activity diagram named *A0* represents the interaction between *System 1* and *System 2*. The initial node represented by the filled dark circle and final node represented by the bulls-eye indicate the start and finish of the activity, respectively. The activity specifies a simple sequence of actions starting with the execution of action *:A1*, and followed by the execution of action *:A2*. The output of *:A1* and the input of *:A2* are represented by rectangles on the action boundary called pins. In addition, the activity partitions labeled *:System 1* and *:System 2* are responsible for performing the actions that are enclosed by the partitions. The action called *:A1* satisfies the requirement *R1.2* which is represented by the *satisfy* relationship.

The action called *:A1* in the activity diagram *A0* is decomposed in the activity diagram called *A1* into actions *:A1.1* and *:A1.2*. These actions are performed by *:Component 1* and *:Component 2,* respectively. The output of the activity *A1* represented by the rectangle on its boundary corresponds to the output pin of action *:A1* in activity *A0.* As indicated in the activity diagrams for *A0* and *A1*, the outputs and inputs are consistent from one level of decomposition to the next.

The block definition diagram is labeled bdd and is often used to describe the hierarchy of a system similar to a parts tree (e.g., equipment tree). A block is used to define a system or component at any level of the system hierarchy. The block definition diagram in the figure shows the block *System Context* composed of *System 1* and *System 2*. *System 1* is further decomposed into *Component 1* and *Component 2*. The *System 1* and *Component 1* blocks each contain a value property that can correspond to a physical or performance characteristic, such as its weight or response time.

The internal block diagram is labeled ibd and shows how the parts of *System 1* are interconnected. The enclosing diagram frame represents *System 1*. The small squares on *System 1* and its parts are called ports and represent their interfaces. *System 1* is also represented by the activity partition in the activity *A0*, and the components are similarly represented by activity partitions in the activity *A1*.

The parametric diagram is labeled par and is used to describe relationships among properties that correspond to an engineering analysis, such as performance, reliability, or mass properties analysis. In this example, the parametric diagram includes a single constraint called *Constraint 1* that corresponds

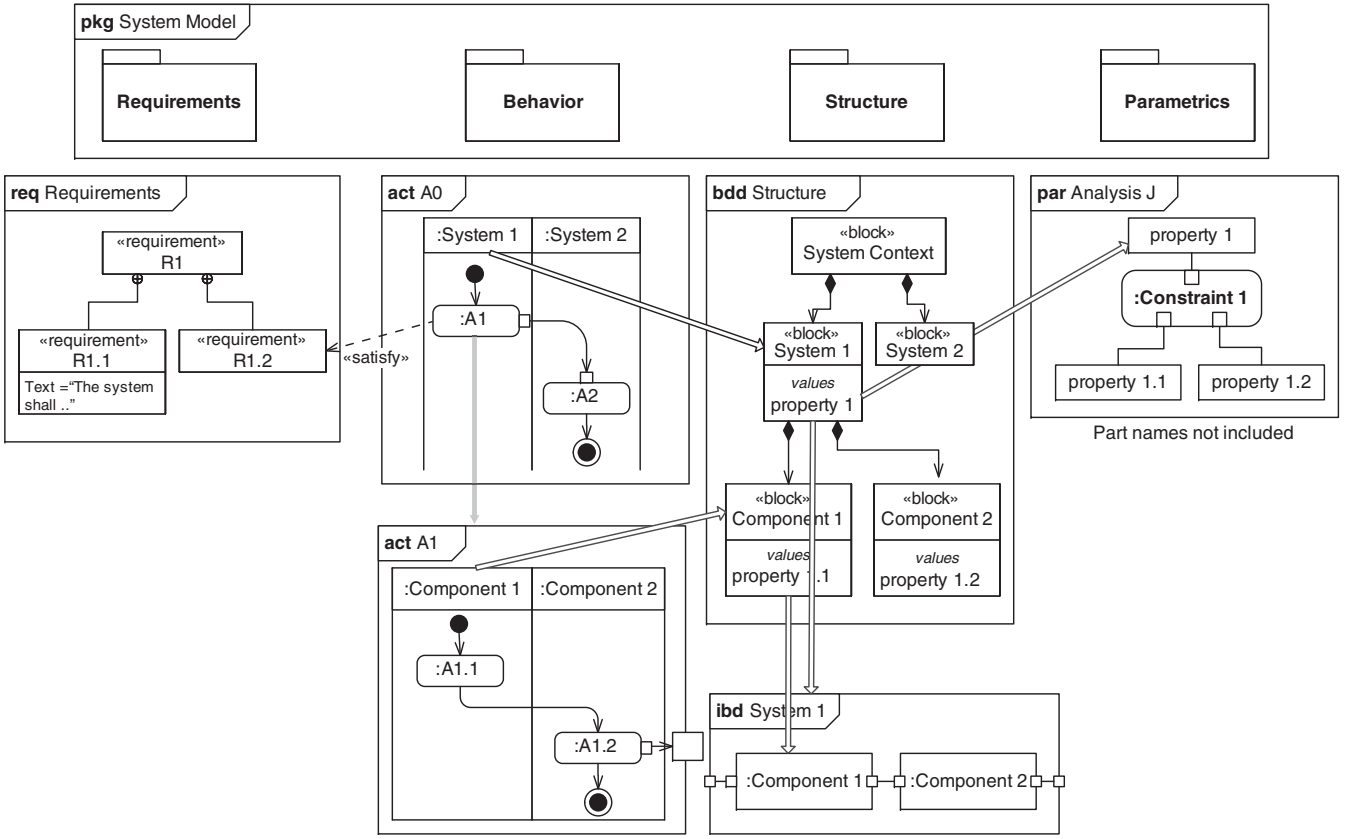


FIGURE 3.3

Simplified diagrams highlighting some of the language features for each kind of diagram in SysML-Lite.

to an equation or set of equations. The small squares flush with the inside of the constraint represent the parameters of the equation. The properties of the system and component blocks can then be bound to the parameters to establish an equality relationship. In this way, a particular analysis can be aligned with the properties of the system design. Often, a single constraint is used to represent a particular analysis, and the parameters represent the inputs and outputs of the analysis.

In the above diagrams, only a small subset of the SysML language features are illustrated to indicate some of the key constructs used to model systems. The following simplified model of an air compressor illustrates how SysML-Lite diagrams and language features can be applied.

Note that some of the names include a colon (:). This is described in Chapter 4, Section 4.3.12, and is further described in Chapter 7, Section 7.3.1.

3.3.2 SysML-Lite Air Compressor Example

The following is an example of using SysML-Lite to model an air compressor that is used to power an air tool. This model is highly simplified for the purposes of illustration, and includes the same type of diagrams that were shown in Figure 3.3.

Figure 3.4 shows the package diagram for the *Air Compressor Model* and includes packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. The model organization follows a similar pattern as described in the section on SysML-Lite above and shown in Figure 3.3.

The *Requirements* package contains a set of requirements that would generally be found in a system specification for the air compressor. The requirements are captured in the requirements diagram in Figure 3.5. The top level requirement called *Air Compressor Specification* contains a functional requirement to compress air, performance requirements that specify the maximum pressure and maximum flow rate, a requirement to specify storage capacity, power requirements to specify the source power needed to compress the air, and reliability and portability requirements. The text for the *Storage Capacity* requirement appears in the diagram, whereas the text for the other requirements are not displayed to reduce the clutter.

The *Behavior* package contains an activity diagram, shown in Figure 3.6, called *Operate Air Tool* that specifies how the *Air Compressor* interacts with the external systems, including the *Air Tool,* the

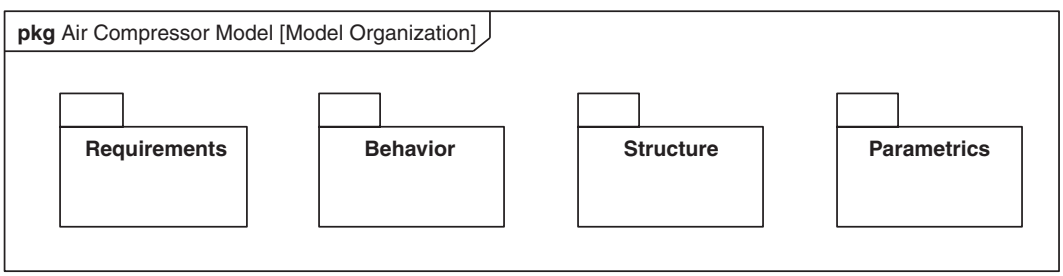


FIGURE 3.4

This package diagram is used to organize the *Air Compressor Model* into packages for *Requirements*, *Structure*, *Behavior*, and *Parametrics*. Each package contains model elements that can be related to model elements in other packages.

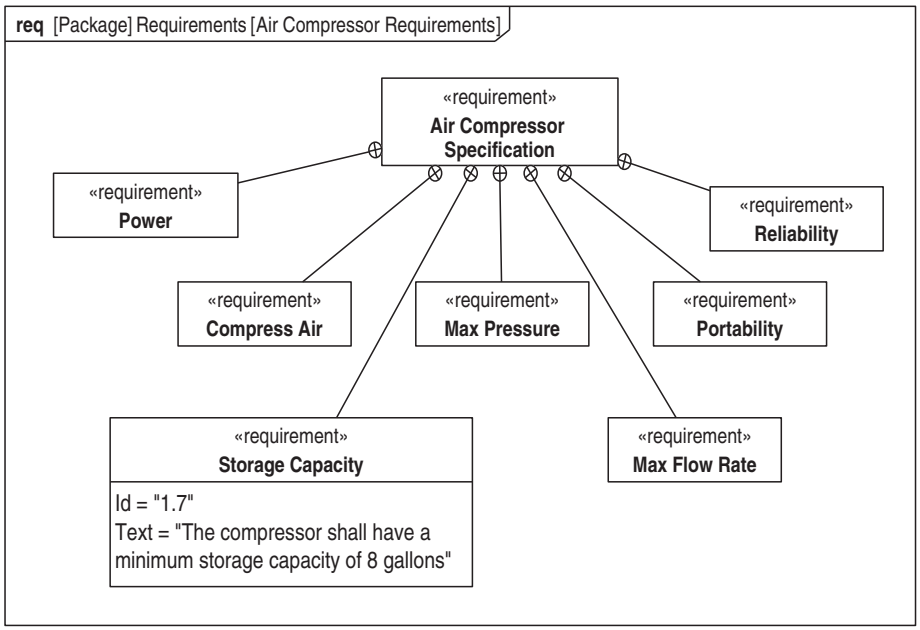


FIGURE 3.5

This requirement diagram represents the requirements contained in the *Requirements* package to specify the *Air Compressor*. Each requirement can include the requirements text that is typically found in a specification document.

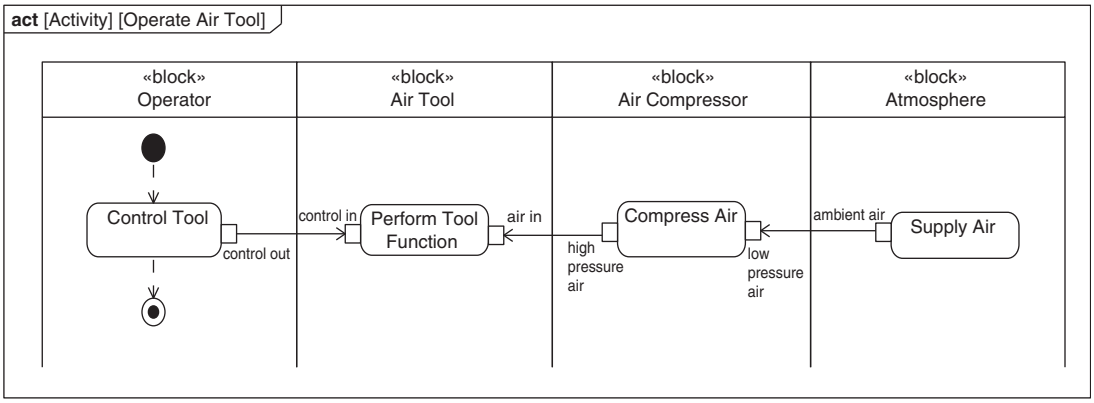


FIGURE 3.6

This activity diagram specifies the interaction between the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* to execute the *Operate Air Tool* activity.

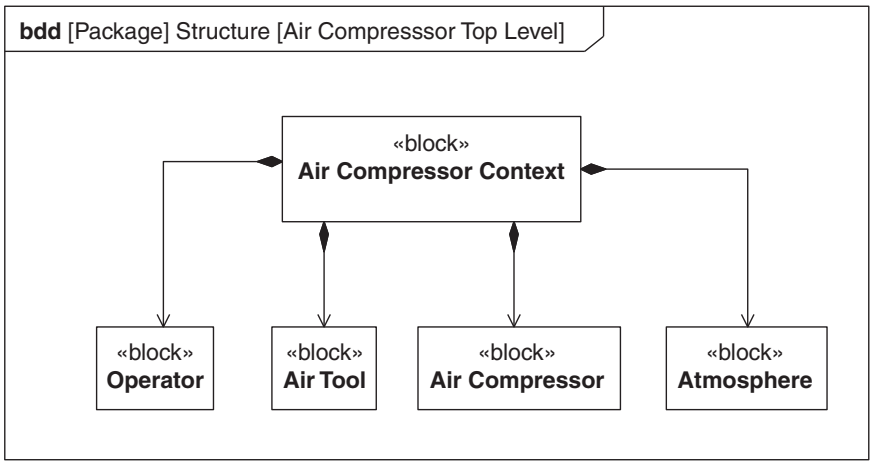


FIGURE 3.7

This block definition diagram represents the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* as blocks. The *Air Compressor Context* block sets the context for the *Air Compressor* and its external environment.

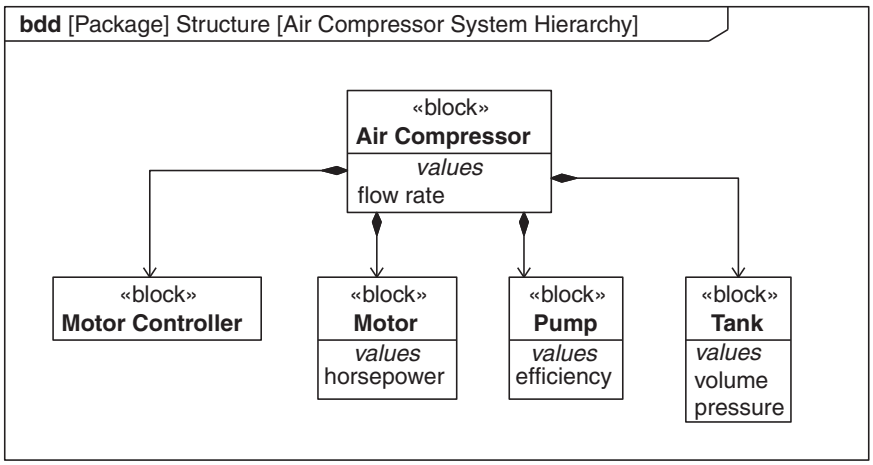


FIGURE 3.8

This block definition diagram represents the *Air Compressor* and its components. The *Air Compressor* block is the same block that is in Figure 3.7.

Atmosphere, and indirectly with the *Operator*, which are represented as activity partitions. The *Air Compressor* performs the function (i.e., action) called *Compress Air*, which has a *low pressure air* input and a *high pressure air* output. The activity begins at the initial node (i.e., dark-filled circle), and then the *Operator* executes the *Control Tool* action. The activity completes its execution at the activity final node (i.e., bulls-eye symbol), after the *Control Tool* action completes execution. The *Compress Air* action is further decomposed in Figure 3.9.

The *Structure* package contains the blocks represented in the block definition diagrams in Figure 3.7 and Figure 3.8. The block definition diagram in Figure 3.7 called *Air Compressor Top Level* includes a block called the *Air Compressor Context* that is composed of the *Air Compressor* and the blocks representing the user, external system, and the physical environment. In this example, the user is the *Operator*, the external system is the *Air Tool*, and physical environment is the the *Atmosphere*. The block definition diagram in Figure 3.8 is called *Air Compressor System Hierarchy.* The *Air Compressor* block in this figure is the same block that is shown in Figure 3.7, but this figure shows that the *Air Compressor* block is composed of components that include the *Motor Controller*, *Motor*, *Pump*, and *Tank*. The *Air Compressor*, *Motor, Tank*, and *Pump* all include value properties that are used to analyze the flow rate requirements.

The activity diagram in Figure 3.9 decomposes the action called *Compress Air* from Figure 3.6 to specify how the components of the *Air Compressor* interact to compress the air. The activity partitions in the activity diagram represent the components of the air compressor. The *Motor Controller* includes actions to *Sense Pressure* and *Control Motor.* The *Motor* performs the action to *Generate Torque*, the *Pump* performs the action to *Pump Air*, and the *Tank* performs the action to *Store Air.* The *low pressure air* input and *high pressure air* output are consistent with the input and output of the *Compress Air* action in Figure 3.6. This activity is contained in the *Behavior* package along with the *Operate Air Tool* activity.

The internal block diagram called *Interconnection* in Figure 3.10 shows how the components of the *Air Compressor* from Figure 3.8 are interconnected. The diagram frame represents the *Air Compressor* block and the ports on the diagram frame represent the external interfaces of the *Air Compressor.* The component parts shown on the internal block diagram are contained in the *Structure* package along with the blocks represented on the block definition diagram.

The block definition diagram called *Analysis Context* in Figure 3.11 is used to define the context for performing the flow rate analysis. In particular, it includes a block called *Flow Rate Analysis* that is composed of a constraint block called *Flow Rate Equations*. This constraint block defines the parameters of the equation and the equations, but the equations are not specified at this time. The *Flow Rate Analysis* block also refers to the *Air Compressor Context* block from Figure 3.7 which is the subject of the analysis. Defining the *Analysis Context* enables a parametric diagram to be created for the *Flow Rate Analysis* block as shown in Figure 3.12. The diagram shows the value properties of the *Air Compressor* and its parts that include *flow rate*, tank *volume* and *pressure*, motor *horsepower*, and pump *efficiency*, and how they are bound to the parameters of the *Flow Rate Equations*. The flow rate analysis equations can be solved by an analysis tool, to determine the property values for the *Air Compressor* and its parts. The analysis context pattern is described further in Chapter 8, Section 8.10 and in Chapter 17, Section 17.3.6.

This air compressor example illustrates how a system can be modeled with a subset of SysML diagrams and language features called SysML-Lite. Even a simple model such as this can contain many model elements, and quickly become difficult to manage. A modeling tool is needed to

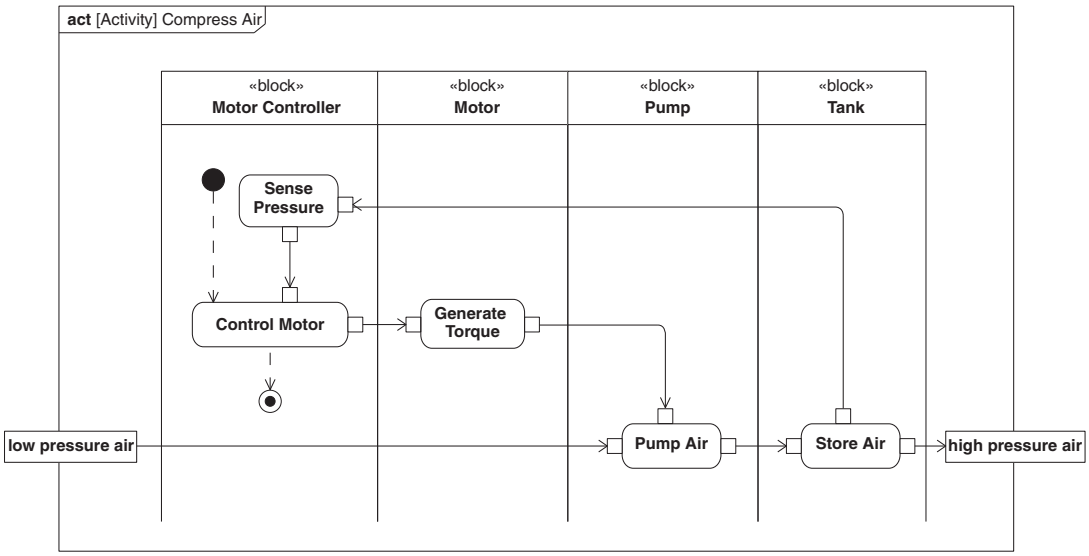


FIGURE 3.9

This activity diagram shows how the components of the *Air Compressor* interact to perform the *:Compress Air* action from Figure 3.6.

efficiently build a model that is self consistent, and to manage complexity. The following section describes how a typical SysML modeling tool is used to build this model.

3.3.3 SysML Modeling Tool Tips

This section provides a brief introduction on how to start modeling with a typical **SysML modeling tool**. The question of how to start modeling often arises when one opens a modeling tool for the first time. Although each tool may have significant differences, the tools typically share much in common from a user interface perspective. As a result, once a modeler learns how to build a SysML model in one tool, it generally takes considerably less time to learn how to model in another tool. Chapter 18 includes a discussion on SysML modeling tools including their role in a typical systems development environment, how to integrate the SysML modeling tool with other tools, and suggested criteria for selecting a SysML modeling tool.

The Tool Interface

The user interface for a typical modeling tool is shown in Figure 3.13, and typically includes a diagram area, a pallet (also known as toolbox), a model browser, and a toolbar. The diagram area is where the diagram appears. The pallet includes diagram elements that are used to create or modify a diagram. The pallet is typically context sensitive such that the diagram elements that appear in the pallet depend on the diagram that is being viewed in the diagram area. For example, if a block definition diagram is being viewed in the diagram area, then the pallet will contain blocks

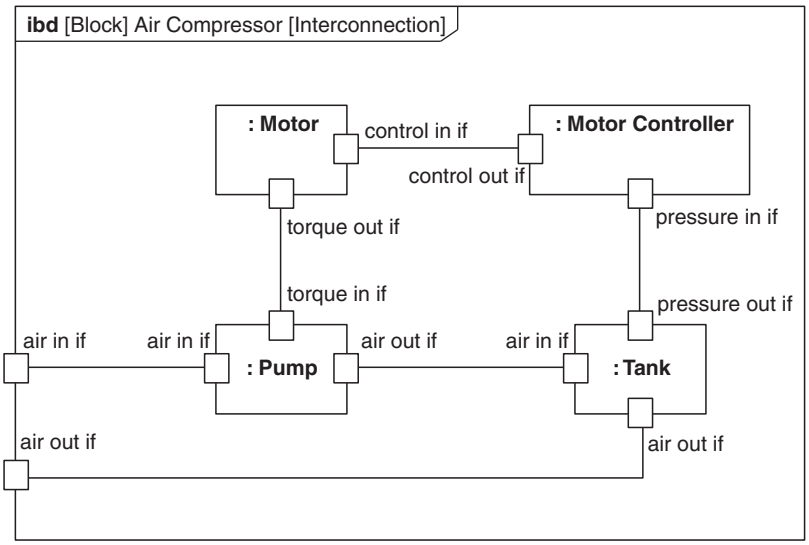


FIGURE 3.10

This internal block diagram shows how the components of the *Air Compressor* are interconnected via their ports, which are used to specify the component interfaces.

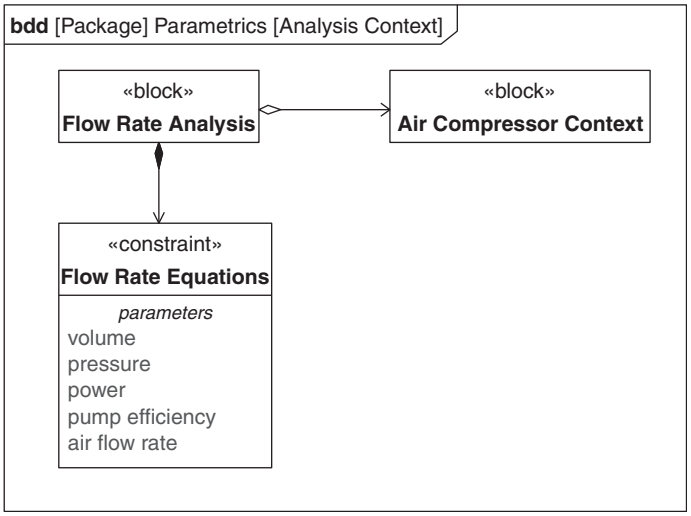


FIGURE 3.11

This block definition diagram is used to specify the *Flow Rate Analysis* in terms of a constraint block that defines the equations and parameters for the analysis (equations not shown), and the *Air Compressor Context* which is the subject of the analysis.

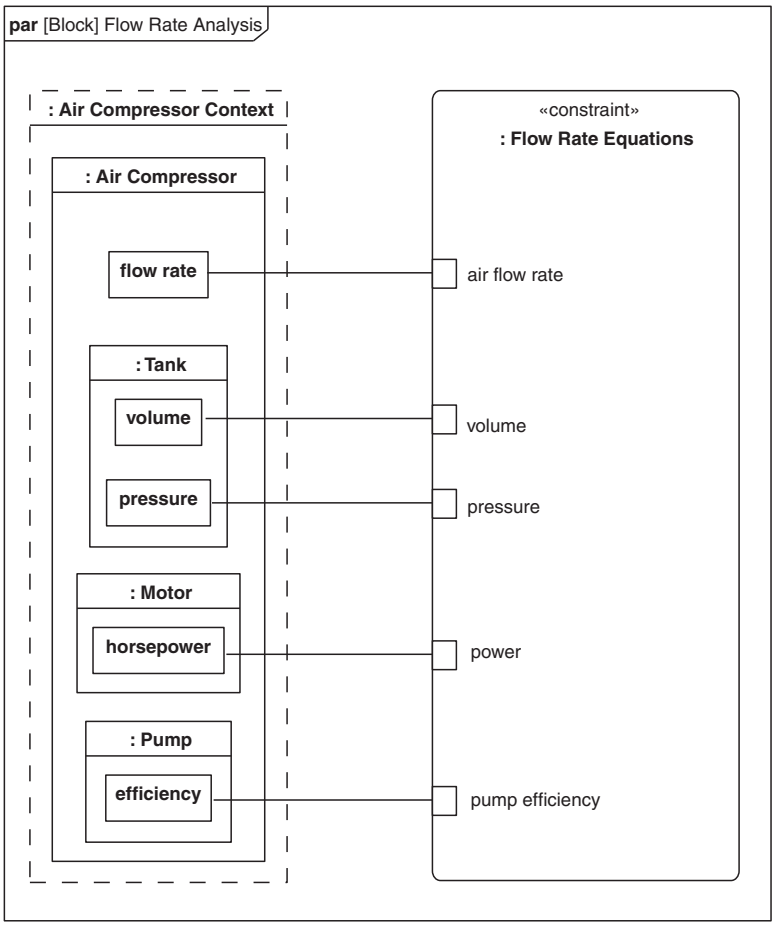


FIGURE 3.12

This parametric diagram represents the *Flow Rate Analysis,* and how the parameters of the equations are bound to the properties of the design. Once captured, this analysis can be provided to an analysis tool to perform the analysis. The equations are not shown in the figure.

and other elements used on a block definition diagram, whereas if an activity diagram is being viewed, the pallet will include actions and other elements used on an activity diagram. The model browser is a third part of the user interface which represents a hierarchical view of the model elements contained in the model. A typical view of the browser shows the model elements grouped into a package hierarchy, where each package appears like a folder that can be expanded to see its content. A package may contain other nested packages. The toolbar contains a set of menu selections that support different operator actions related to file management, editing, viewing, and other actions.

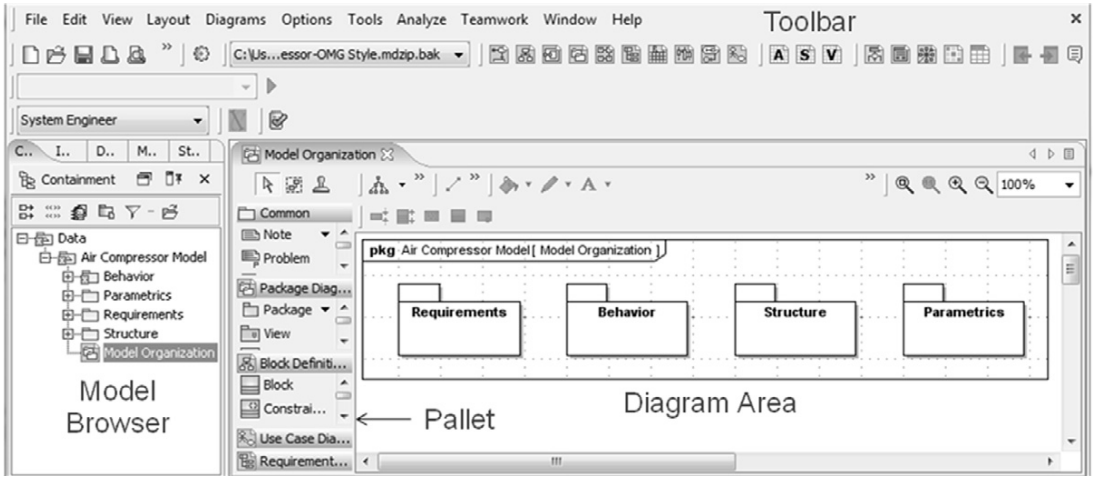


FIGURE 3.13

A typical SysML modeling tool interface consists of a diagram area, a pallet or toolbox, a model browser, and a toolbar. The model browser shows the hierarchy of model elements that are contained in the model.

To create a new diagram, a modeler selects a diagram kind and names the diagram. There are often multiple ways to select a diagram kind, such as from a diagram menu or a diagram icon from the toolbar. The new diagram appears in the diagram area without any content. The diagram header information is visible and includes the diagram kind, the diagram name, and other information about the diagram frame. The modeler can then drag a diagram element from the pallet onto the diagram in the diagram area and name the new element. Once this is done, the corresponding model element appears in the browser. As an alternative, the modeler can add the new model element directly in the browser, and then drag this model element onto the diagram. A model element appears in only one place in the browser, but may appear on zero, one or more diagrams.

A modeling tool provides mechanisms for navigating between the elements on the diagrams and the model elements in the browser. This can be important since a large model may contain hundreds of diagrams, and thousands or hundreds of thousands of model elements. Most tools allow the modeler to select the model element on the diagram and request its location in the browser. A modeler can also select a model element in the browser, and request a list of the diagrams where the model element appears.

The modeling tool allows the modeler to show and hide selected details of the model on any particular diagram. This is important for managing the complexity of the diagrams. The modeler only shows what is considered important to support the purpose of the diagram.

If the modeler wishes to delete a model element from the diagram, the tool may prompt the modeler whether to delete the model element from the diagram only, or to delete the model element from the model as well. A modeler can also choose to delete a model element from the model by selecting the model element in the browser, and then deleting it.

A modeling tool has many other capabilities that enable a modeler to develop and manage a system model. Once the model element is created, the modeler can typically select the model element and open up its specification where details of the model element can be added, modified, or deleted. The modeler can also select a model element on the diagram, and query the modeling tool to show all of the directly related model elements that can appear on that particular kind of diagram.

It is also worth noting that the modeling tool is often used in conjunction with a configuration management tool to put the model under configuration control. This is particularly important when modeling as part of a distributed team where multiple people are working on the same model. In this case, a typical configuration management tool will allow read and/or write privileges to be assigned to a user to control their access to different parts of the model. Once this is done, a modeler with read privileges assigned to a particular part of the model can view that part of the model, and a modeler with write privileges assigned to a particular part of the model can also check out and modify that part of the model.

Chapter 18 describes how the SysML modeling tool integrates into a Systems Development Environment with many other tools including configuration management, requirements management, hardware and software design, and analysis tools.

Building the Model

The following illustrates how to build the *Air Compressor Model* from Section 3.3.2 in a typical modeling tool. Each tool will have its unique user interface, and different modeling guidelines and MBSE methods may suggest different ways to get started. However, the following example provides a representative starting point, which can be further adapted to the specific modeling tool, modeling guidelines, and MBSE method.

The modeler must first install and configure the modeling tool so that it can be used to build a model that is represented in SysML. Many SysML tools also support UML and perhaps other modeling languages, so the modeler may be required to select and apply SysML. Once this is done, a modeler can create a new project and name this project, such as the Air Compressor Project.

As indicated in Figure 3.13, the first step in building the model is to create the top level package in the browser called the *Air Compressor Model.* The modeler can then select this package in the browser, and create nested packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. Alternatively, the modeler can create a new package diagram similar to the one shown in Figure 3.4, by dragging new packages from the pallet onto the diagram and naming them accordingly.

The modeler can then select the *Requirements* package in the browser, and create a new requirements diagram and name it *Air Compressor Requirements*. Once the diagram appears in the diagram area, the modeler can drag new requirements from the pallet onto the diagram and name them to correspond to the requirements in Figure 3.5. The relationship between model elements can then be defined using the kind of relationships shown in the pallet. In the case of requirements, a parent and child requirement can be related by connecting the parent requirement to each child requirement with the cross hair symbol at the parent requirement end.

The modeler next creates the top level activity diagram *Operate Air Tool* shown in Figure 3.6. This is done by selecting the *Behavior* package, and creating a new activity diagram, and naming the

diagram *Operate Air Tool*. The modeler may drag actions from the pallet onto the activity diagram, along with the initial and final nodes, and connect the actions with the appropriate flow. The control flow is used to connect the initial node to *Control Tool*, and another control flow connects *Control Tool* to the activity final node. The object flows connect the inputs and outputs for each of the actions. The activity partitions can be added after the next step in the process.

The modeler next creates the block definition diagram for the *Air Compressor Context* shown in Figure 3.7. This is accomplished by selecting the *Structure* package in the browser and creating a new block definition diagram, and naming it *Air Compressor Top Level*. A new block can be dragged from the pallet onto the diagram and called *Air Compressor Context* block. The other blocks can then be defined similarly. The composition relationship between the *Air Compressor Context* block and the other blocks can be established in a similar way as described for the requirements diagram but using the composition relationship designated by the black diamond on one end of the line.

Once the blocks are defined, the activity partitions (i.e., swim lanes) in the activity diagram in Figure 3.6 can be defined to represent these blocks. This activity diagram specifies the interaction between the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* to execute the *Operate Air Tool* activity. This is accomplished by selecting the previously created activity diagram, *Operate Air Tool,* to view in the diagram area. The modeler then drags the activity partitions from the pallet onto the diagram. In order to represent an activity partition by a particular block, the modeler typically opens the activity partition specification, and then selects the particular block to represent the partition. Each action is then placed within the activity partition corresponding to the block that is responsible for performing the action.

The modeler can then decompose the system into its component parts by creating the block definition diagram shown in Figure 3.8. This is done by selecting the *Structure* package, and creating a new block definition diagram, and naming it *Air Compressor System Hierarchy.* New blocks can be dragged from the pallet onto the diagram, and the relationships are established in a similar way as described for the block definition diagram called *Air Compressor Top-Level*. The ports on each of the blocks can then be created by dragging a port from the pallet onto the block, or, alternatively, by selecting a block and opening up its specification, and then adding the ports. In addition, the properties of the block can be created be opening up each block's specification on the diagram or from the browser, adding a new property, and naming it. In this example, both the ports and the properties are included in the model, but not shown on the diagram, in order to further simplify the diagram.

The modeler next creates the activity diagram to show the interaction between the parts of the *Air Compressor* as shown in Figure 3.9. This activity diagram is created in a similar way as the previous activity diagram *Operate Air Tool*. However, this activity is used to decompose the *Compress Air* action that the *Air Compressor* performs in the *Operate Air Tool* activity. The new activity is created by first ensuring the *Compress Air* action is a special type of action called a call behavior action, which then calls the new activity called *Compress Air.* The activity partitions can then be dragged from the pallet onto the diagram, and can now represent the component blocks created in the *Air Compressor System Hierarchy* block definition diagram.

The modeler next creates the internal block diagram shown in Figure 3.10. This is accomplished by selecting the *Air Compressor* block in the *Structure* package in the browser, and creating

a new internal block diagram. When the composition relationships were previously created between the *Air Compressor* and its component blocks, the tool should create new model elements in the browser under the *Air Compressor* block. These elements are called parts, and are used in the internal block diagram for the *Air Compressor.* The parts of the *Air Compressor* block are dragged from the browser onto the internal block diagram, and then connected to one another via their ports. The ports on the parts may not be visible on the diagram. Many tools require the modeler to select the part, and select a menu item to display the ports. The ports can be connected to one another once the ports are visible on the diagram. A modeler may also connect the parts without ports, and add ports later if desired.

The modeler next creates the block definition diagram in Figure 3.11 to specify the constraints used in the parametric diagram. This is done by selecting the *Parametrics* package in the browser, and creating a new block definition diagram and naming the diagram *Analysis Context*. The *Flow Rate Analysis* block is created, and the *Air Compressor Context* block that is contained in the *Structure* package is dragged onto the diagram and referenced (i.e., white diamond aggregation) by the *Flow Rate Analysis* block. A new constraint block called *Flow Rate Equations* is then created, and related to the *Flow Rate Analysis* block with a composition relationship. The parameters of the constraint block are then defined in a similar way as the properties of blocks described earlier. The equations can be specified as part of the constraint block.

The modeler next creates the parametric diagram shown in Figure 3.12. The constraint property which is typed by the *Flow Rate Equations* constraint block and a part which is typed by the *Air Compressor Context* block are dragged from the browser onto the diagram. The *Air Compressor Context* is selected on the diagram, and its nested parts and value properties are displayed on the diagram. Different tools accomplish this in different ways. Once this is done, the value properties contained in the *Air Compressor*, *Tank*, *Motor,* and *Pump* can be connected to the parameters of the *Flow Rate Equations* constraint property.

Creating this example in the modeling tool is an important first step to learning how to model. Once this is understood, one can learn additional SysML language features, and explore additional tool capabilities such as documentation generation, tabular representations, diagram layout functions, etc. The automobile example in Chapter 4 introduces the remaining three (3) SysML diagrams and additional language features that can serve as a next step in the learning process.

3.4 A SIMPLIFIED MBSE METHOD

In addition to learning the modeling language and a tool, a modeler must apply a disciplined model-based systems engineering (MBSE) method to adhere to sound systems engineering practice and build quality system models. SysML provides a means to capture the system modeling information without imposing a specific MBSE method. A selected method determines which modeling activities are performed, the ordering of the activities, and the types of modeling artifacts used to represent the system. For example, traditional structured analysis methods can be used to decompose the functions and then allocate the functions to components. Alternatively, one can apply a use case driven approach that derives functionality based on scenario analysis and the associated interactions among the parts. The two methods may involve different activities and produce different combinations of diagrams to represent the system specification and design. Several MBSE methods are documented in the Survey

of Model-based Systems Engineering Methodologies [5]. Chapters 16 and 17 provide two examples with different MBSE methods.

The top level activities for a simplified MBSE method are highlighted in Figure 3.14. The activities are consistent with the systems engineering process introduced in Chapter 1, Section 1.2. The method also represents a simplified variant of the object-oriented systems engineering method (OOSEM) that is described in detail as it is applied to the residential security example in Chapter 17. This method includes one or more iterations of the following activities to specify and design the system:

- *Organize the Model*
  - Define the package diagram for the system model
- *Analyze Stakeholder Needs* to understand the problem to be solved, the goals the system is intended to support, and the effectiveness measures needed to evaluate how well the system supports the goals
  - Identify the stakeholders and the problems to be addressed
  - Define the domain model to identify the system and external systems and users
  - Define the top level use cases to represent the goals the system is intended to support
  - Define the effectiveness measures that can be used to quantify the value of a proposed solution
- *Specify System Requirements* including the required system functionality, interfaces, physical and performance characteristics, and other quality characteristics to support the goals and effectiveness measures
  - Capture text-based requirements in a requirements diagram that support the system goals and effectiveness measures
  - Model each use case scenario (e.g., activity diagram) to specify the system behavior requirements
  - Create the system context diagram (internal block diagram) to specify the system external interfaces
- *Synthesize Alternative System Solutions* by partitioning the system design into components that can satisfy the system requirements
  - Decompose the system using the block definition diagram
  - Define the interaction among the parts using activity diagrams
  - Define the interconnection among the parts using the internal block diagram
- *Perform Trade-off Analysis* to evaluate and select a preferred solution that satisfies the system requirements and maximizes value based on the effectiveness measures
  - Capture the analysis context to identify the analysis to be performed such as performance, mass properties, reliability, cost, and other critical properties
  - Capture each analysis as a parametric diagram
  - Perform the engineering analysis to determine the values of the system properties (Note: the analysis is performed in engineering analysis tools)
- *Maintain Requirements Traceability* to ensure the proposed solution satisfies the system requirements and associated stakeholder needs
  - Capture the traceability between the system requirements and the stakeholder needs
  - Show how the system design satisfies the system requirements
  - Identify test cases needed to verify the system requirements and capture the verification results

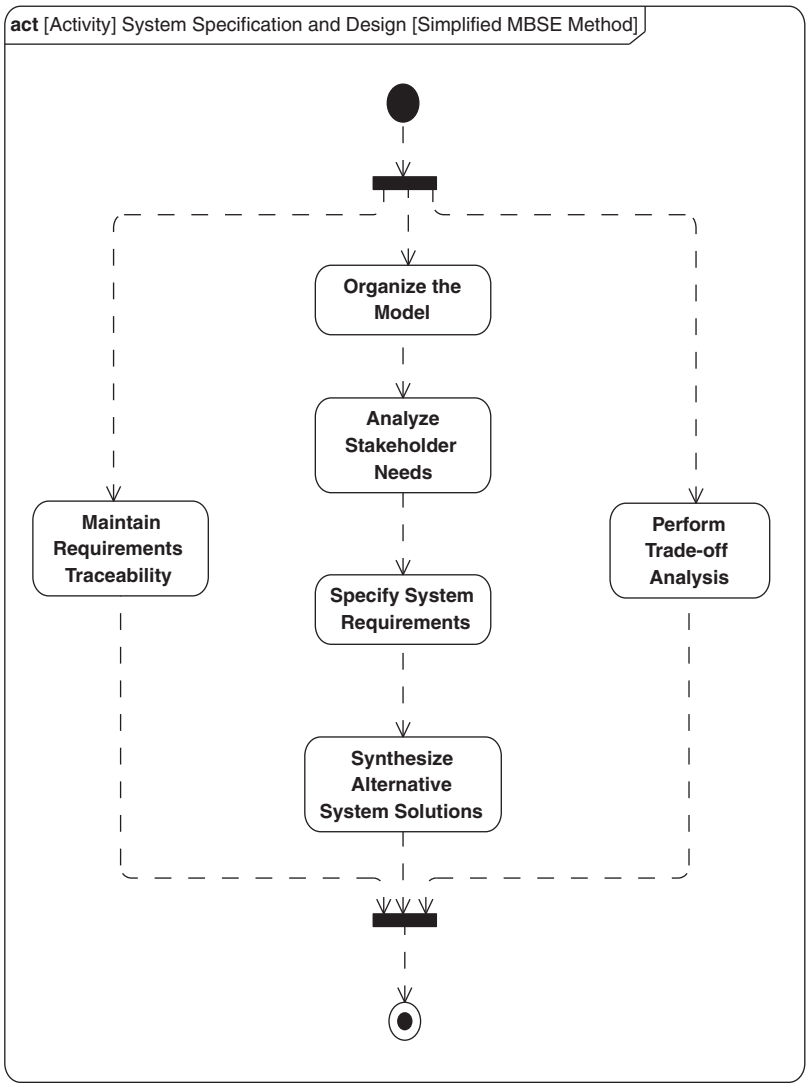


FIGURE 3.14

A simplified MBSE method that is consistent with the systems engineering process described in Chapter 1, Section 1.2. The method is used to produce the modeling artifacts that constitute the system model.

Other systems engineering management activities, such as planning, assessment, risk management, and configuration management are performed in conjunction with the modeling activities described above. Detailed examples of how SysML can be used to support a functional analysis and allocation method and the object-oriented systems engineering method (OOSEM) are included in the modeling

examples in Part III Chapters 16 and 17, respectively. A simplified example is described in the next chapter, which illustrates some of the model-based artifacts that are generated when applying a typical MBSE method.

3.5 THE LEARNING CURVE FOR SysML AND MBSE

Learning SysML and MBSE requires a commitment similar to what is expected of learning modeling for mechanical, electrical, software, and other technical disciplines. The challenges to learning SysML and MBSE have some additional factors that contribute to its learning curve. In particular, a major focus for model-based systems engineering approaches is the ability to understand a system from multiple perspectives, and to ensure integration across the different perspectives. In SysML, the system requirements, behavior, structure, and parametrics each represents different aspects of the system that need to be understood individually and together.

Each of the individual perspectives introduces its own complexity. For example, the modeler may represent behavior in activity diagrams to precisely specify how a system responds to a stimulus. This involves specifying the details of how the system executes each use case scenario. These activity diagrams may be integrated into a composite system behavior that is captured in a state machine diagram. The process for representing detailed behavior and integrating different behavior formalisms can be quite complex.

As stated above, the modeler must maintain consistency from one perspective to another. SysML is often used to represent hierarchies for requirements, behavior, structure, and parametrics. A consistent model of a system must ensure consistency between the model elements in each hierarchy. Some of these relationships were highlighted in the examples in Sections 3.3.1 and 3.3.2. Additional discipline-specific views may cross cut the requirements, behavior, structure, and parametrics perspectives, such as a reliability view, security view, or manufacturing view. Again, this introduces complexity to system modeling and MBSE.

Another aspect of complexity is that an effective MBSE approach not only requires a language such as SysML to represent the system, but also a method that defines the activities and artifacts, and a tool to implement the modeling language and method. The language, method, and tool each introduce their own concepts, and must be learned to master model-based systems engineering. This skill must then be applied to a particular domain, such as designing aircraft, automobiles, telecommunication systems, medical devices, and others.

Additional modeling challenges are associated with scaling the modeling effort to larger projects, and in the context of a diverse development environment. The challenges of managing the model come into play. There may be multiple modelers in multiple locations. Disciplined processes and associated tools must be put in place to manage changes to models. There are also many different types of models involved in an MBSE effort beyond the SysML model, such as a multitude of analysis models, hardware models, and software models. The integration among the different models and tools, and other engineering artifacts is another challenge associated with MBSE.

Model-based systems engineering formalizes the practice of how systems engineering is performed. The complexity and associated challenges for learning MBSE reflect the inherent complexity and challenges of applying systems engineering to the development of complex systems. Some of this complexity was highlighted in the automobile design example in Chapter 1, Section 1.3 independent of

the MBSE approach. When starting out on the MBSE journey, it is important to set expectations for the challenges of learning MBSE and how to apply it to the domain of interest. However, in addition to reaping the potential benefits of MBSE described in Chapter 2, embracing these challenges and becoming proficient in SysML and MBSE can provide a deeper understanding of systems and systems engineering concepts.

3.6 SUMMARY

SysML is a general-purpose graphical language for modeling systems that may include hardware, software, data, people, facilities, and other elements within the physical environment. The language supports modeling of requirements, structure, behavior, and parametrics to provide a robust description of a system, its components, and its environment.

The language includes nine diagram kinds each with and many features. The semantics of the language enable a modeler to develop an integrated model of a system, where each kind of diagram can represent a different view of the system being modeled. The model elements on one diagram can be related to model elements on other diagrams. The diagrams enable capturing the information in a model repository, and viewing the information from the repository, to help specify, design, analyze, and verify systems. To facilitate the learning process, SysML-Lite is introduced, which includes six of the nine SysML diagrams and a relatively small subset of the language features for each diagram kind. Learning how to model this subset of the language in a modeling tool can provide a sound foundation to build on.

The SysML language is a critical enabler of MBSE. Effective use of the language requires a well-defined MBSE method. SysML can be used with a variety of MBSE methods. This chapter introduced a simplified MBSE method to aid in getting started.

SysML enables representation of a system from multiple perspectives. Each of the individual perspectives may be complex in their own right, but ensuring a consistent model that integrates across the different perspectives introduces additional challenges to learning SysML and MBSE. When learning SysML as part of an overall MBSE approach, the process, methods, and tools introduce their own concepts and complexity. Using SysML in support of MBSE formalizes the practice for how systems engineering is performed. Ultimately, the challenges of SysML and MBSE reflect the inherent complexities of applying systems engineering to developing complex systems. The learning expectations should be set accordingly.

3.7 QUESTIONS

1. What are five aspects of a system that SysML can represent?
2. What is a package diagram used for?
3. What is a requirement diagram used for?
4. What is an activity diagram used for?
5. What is the block definition diagram used for?
6. What is an internal block diagram used for?
7. What is a parametric diagram used for?
8. What are some of the common elements of the user interface of a typical SysML modeling tool?

9. Which element of the user interface reflects the model repository?
10. What is the purpose of applying an MBSE method?
11. What are the primary activities of the simplified MBSE method?

Discussion Topics

What are some factors that contribute to the challenges of learning SysML and MBSE, and how do they relate to the general challenges of learning systems engineering?